# Grounding FO(ID) with Bounds

Johan Wittocx* and Marc Denecker

Department of Computer Science, K.U. Leuven
{johan.wittocx,marc.denecker}@cs.kuleuven.be

**Abstract.** Grounding is the task of reducing a given first-order theory $T$ and finite domain to an equivalent propositional theory. It is used as preprocessing step in many logic-based reasoning systems. In this paper, we present a method to improve grounding for FO(ID), the extension of first-order logic with inductive definitions. The method consists of computing *bounds* for subformulas of $T$, indicating for which part of the given domain, the truth value of their subformula is the same in every model of $T$. Bounds can be used to efficiently produce compact groundings. We present both theoretical results and experiments to support this claim.

## 1 Introduction

Grounding, or propositionalization, is the task of reducing a first-order theory and a finite domain to an equivalent propositional theory, called *a grounding*. Grounding is used as a preprocessing phase in many logic based systems such as finite model generators for (extensions of) classical logic [2, 6, 12, 14, 19, 22] and in Answer Set Programming [8, 15, 18], planning [9], and relational data mining [10] systems. In this paper[1], we focus on finite model generation for first-order logic (FO) and FO(ID), an extension of FO with inductive definitions.

A basic (naive) grounding method is by instantiating variables by domain elements. Grounding in this way is polynomial in the size of the domain but exponential in the quantifier rank of formulas in the input theory, and may easily produce propositional theories of unwieldy size. Several methods have been developed to efficiently produce smaller groundings. There are two main categories of such methods. In the first, the input theory $T$ is rewritten such that the quantifier rank of formulas in $T$ decreases (see, e.g., [16, 17]). The methods of the second category are applicable when the truth value of some of the atoms occurring in the basic grounding of $T$ are given. By replacing these atoms by their truth value and then simplifying the result, a smaller grounding is obtained. In this paper, we present a method of the second category. To explain the intuition underlying our method, consider the following model generation problem.

* Research assistant of the Fund for Scientific Research - Flanders (FWO-Vlaanderen)
[1] This paper is a summary of [25] and of Chapter 5 of [21].

*Example 1.* Let $T_1$ be the FO theory over the vocabulary $\{Edge, Sub\}$, consisting of the two sentences

$$\forall u \forall v \ (Sub(u,v) \supset Edge(u,v)) \tag{1}$$

$$\forall x \forall y \forall z \ (Sub(x,y) \wedge Sub(x,z) \supset y = z). \tag{2}$$

$T_1$ expresses that $Sub$ is a subgraph of $Edge$ with at most one outgoing edge in each vertex. Computing such a subgraph of a given graph $G = \langle V, E \rangle$ can be cast as the model generation problem with input theory $T_1$, domain $V$, and fixed truth value *true*, respectively *false*, for every atom $Edge(v_1, v_2)$ such that $(v_1, v_2) \in E$, respectively $(v_1, v_2) \in V^2 \setminus E$. The basic grounding algorithm would produce $|V|^2$ instantiations of (1) and $|V|^3$ of (2). Replacing the atoms with a fixed truth value, i.e., atoms of the form $Edge(v_1, v_2)$ and $v_1 = v_2$, eliminates $|E|$ instantiations of (1) and $|V|$ instantiations of (2).

A typical way to reduce the grounding size even further is by adding redundant information to the input theory. E.g.,

$$\forall x \forall y \forall z \ (Edge(x,y) \wedge Sub(x,y) \wedge Edge(x,z) \wedge Sub(x,z) \supset y = z) \tag{3}$$

is equivalent to (2) given (1), but its grounding (after simplification!) is potentially a lot smaller. For example, if $G$ itself has at most one outgoing edge per vertex, then the grounding of (3) would reduce to the empty theory. This illustrates how adding redundant information may sometimes dramatically reduce the grounding size. However, manually adding redundancy to formulas has its disadvantages: it leads to more complex and hence, less readable theories. Worse, it might introduce errors and it requires a good understanding of the used grounder, since what information is beneficial to add and where, depends on the grounder. Also, a human developer could easily miss useful information.

The above motivates a study of automated methods for deriving redundant information and of principled ways of adding it to formulas. In [23], we presented such methods for first-order logic. Now, we extend these methods to FO(ID) and give details about the implementation of a grounder using redundant information. Experiments at the end of the paper show the impact of using redundant information on several benchmark problems.

## 2 Preliminaries

### 2.1 FO and FO(ID)

We assume that the reader is familiar with classical first-order logic (FO). We introduce the notations and conventions used in this paper. Next, we present FO(ID), an extension of FO with inductive definitions. To facilitate the presentation, we only consider function-free FO and FO(ID) in this paper.

A *vocabulary* $\Sigma$ consists of variables, constants and predicate symbols. Variables and constants are denoted by lowercase letters, predicate symbols by uppercase letters. Sets and tuples of variables are denoted by $\overline{x}, \overline{y}, \ldots$. For a formula $\varphi$, we often write $\varphi[\overline{x}]$, to indicate that $\overline{x}$ are its free variables.

A $\Sigma$-*structure* $I$ consists of a domain $D$ and an assignment of a relation $P^I \subseteq D^n$ to every $n$-ary predicate symbol $P \in \Sigma$ and a domain element $c^I$ to every constant $c \in \Sigma$. A structure is *finite* if its domain is finite. The restriction of a $\Sigma$-structure $I$ to a vocabulary $\sigma \subseteq \Sigma$ is denoted by $I|_\sigma$.

In the rest of this paper, we assume all structures are over the same finite domain $D$. Also, we assume that every vocabulary contains a constant $\mathbf{d}$ for every domain element $d \in D$ and that $\mathbf{d}^I = d$ in every structure $I$. These constants are called *domain constants*. Abusing notation, we make no distinction between domain constants and corresponding domain elements in the rest of the paper. If $d$ is a domain constant and $x$ a variable, then $\varphi[x/d]$ denotes the result of replacing every free occurrence of $x$ in $\varphi$ by $d$. This notation is extended to tuples of variables and domain constants of the same length. To facilitate the presentation, we assume the domain constants are the only constants that occur in a vocabulary.

The satisfaction relation $\models$ is defined as usual (see, e.g., [7]).

The logic FO(ID) [3, 4] is an extension of FO with a construct to represent some of the most common types of inductive definitions: monotone, non-monotone (e.g., induction over a well-founded order), and iterated inductive definitions. Such definitions have many applications in real-life computational problems, e.g., in planning problems or problems about dynamic systems.

In FO(ID) a *definition* $\Delta$ is a finite set of rules of the form $\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi[\overline{y}])$, where $P$ is a predicate symbol, $\varphi$ an FO formula, and $\overline{y} \subseteq \overline{x}$. $P(\overline{x})$ is called the *head* of the rule and $\varphi$ the *body*. We denote by $\mathrm{Def}(\Delta)$ the set of predicates that occur in the head of a rule of a definition $\Delta$. These predicates are called the *defined predicates* of $\Delta$. All other symbols are called the *open symbols* of $\Delta$. The set of open symbols is denoted by $\mathrm{Open}(\Delta)$.

A structure $I$ satisfies a definition $\Delta$, denoted $I \models \Delta$, if $I|_{\mathrm{Def}(\Delta)}$ is the well-founded model [20] of $\Delta$, computed in terms of $I|_{\mathrm{Open}(\Delta)}$. As argued in, e.g., [4], the well-founded semantics is used because it correctly formalizes the semantics of the above mentioned types of inductive definitions.

*Example 2.* Definition $\Delta_1$ defines $TC$ to be the transitive closure of relation $R$.

$$\Delta_1 = \left\{ \begin{array}{l} TC(x,y) \leftarrow R(x,y), \\ TC(x,y) \leftarrow \exists z\ (TC(x,z) \wedge TC(z,y)) \end{array} \right\}$$

In general, the well-founded model is a three-valued structure. A definition $\Delta$ is *total* if for every $\mathrm{Open}(\Delta)$-structure $I$, the well-founded model of $\Delta$ in terms of $I$ is two-valued. In [4], it was argued that total definitions correspond to well-formed definitions. If a definition is not total, this normally indicates an error. Hence all definitions that occur in practice are total. Although it is in general undecidable whether a definition is total, there are several broad and easily recognizable classes of total definitions. For example, all monotone and stratified definitions are total.

An FO(ID) theory $T$ is a finite set of FO sentences and definitions. A structure $I$ satisfies $T$ if it satisfies all definitions and sentences in $T$. Observe that an FO(ID) theory has the appearance of an FO theory augmented with a collection

of *logic programs*. As shown in [5], this entails that FO(ID)'s definitions can not only be used to represent mathematical concepts, but also for the sort of common sense knowledge that is often represented by logic programs, such as (local forms of) the closed world assumption, inheritance, exceptions, defaults, causality, etc. A formula $\varphi$ is a *subformula* of an FO(ID) theory $T$ if it is a subformula of a sentence in $T$ or a subformula of a rule body in a definition of $T$. In particular, heads of rules in definitions of $T$ are not considered to be subformulas of $T$.

### 2.2 Model Expansion and Grounding

A *grounding* of an FO theory $T$ is an "equivalent" propositional theory $T_g$. The notion of equivalence depends on the application one has in mind. In this paper, we consider the application of finite model generation, in a context where an interpretation for some symbols is given. This setting is called *model expansion*. Model expansion for a logic $\mathcal{L}$, denoted MX($\mathcal{L}$) is defined as follows.

**Definition 1.** *Let $T$ be an $\mathcal{L}$-theory over a vocabulary $\Sigma$ and $\sigma$ a subvocabulary of $\Sigma$. The* model expansion search problem $\mathrm{MX}^{\sigma}_{T}$ *is the problem of computing for a given $\sigma$-structure $I_\sigma$ with finite domain $D$, a $\Sigma$-structure $M$ with domain $D$ such that $M \models T$ and $M|_\sigma = I_\sigma$.*

The vocabulary $\sigma$ is called the *input vocabulary* of the problem, $\Sigma \setminus \sigma$ the *expansion vocabulary*. $I_\sigma$ is called the *input structure*. We denote the set of solutions of $\mathrm{MX}^{\sigma}_{T}$ with input $I_\sigma$ by $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$.

MX problems can be solved by creating an appropriate grounding $T_g$ of $T$ using $I_\sigma$ and subsequently calling a propositional model generator for input $T_g$. To this end, a one-to-one correspondence between the models of $T_g$ and the models of $T$ expanding $I_\sigma$ is required, such as $I_\sigma$-equivalence:

**Definition 2.** *Two theories $T_1$ and $T_2$ over $\Sigma$ are* $I_\sigma$-equivalent *if $\mathrm{MX}^{\sigma}_{T_1}(I_\sigma) = \mathrm{MX}^{\sigma}_{T_2}(I_\sigma)$, where $\sigma$ is the vocabulary of $I_\sigma$.*

We now define grounding for MX(FO(ID)) formally. Let $T$ be an FO(ID) theory and $D$ a domain. We say that $T$ is in *ground normal form (GNF) over* $D$ if it contains no variables and no quantifiers. I.e., all its atomic formulas are of the form $P(d_1, \ldots, d_n)$, where $d_1, \ldots, d_n \in D$. A GNF theory $T$ is essentially propositional: to transform $T$ to an equivalent propositional theory, it suffices to introduce a propositional variable for each of the atoms $P(\overline{d})$ and replace in $T$ all atoms by their corresponding variables.

**Definition 3.** *A* grounding for $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$ *is a GNF theory $T_g$ over $\Sigma$ such that $T$ and $T_g$ are $I_\sigma$-equivalent. The theory $T_g$ is* reduced *if it does not contain symbols of $\sigma$.*

For the rest of this paper, let $T$ be an FO(ID) theory over $\Sigma$, $\sigma \subseteq \Sigma$, and $I_\sigma$ an $\sigma$-structure with finite domain $D$. Also, we assume without loss of generality [11] that none of the predicates in $\sigma$ is defined by a definition in $T$. There exists a straightforward grounding for $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$, called the *full grounding*:

**Definition 4.** *The* full grounding $Gr_{full}(\varphi)$ *of a sentence or rule* $\varphi$ *with respect to* $I_\sigma$ *is defined by*

$$Gr_{full}(\varphi) = \begin{cases} \varphi & \text{if } \varphi \text{ is a literal} \\ Gr_{full}(\psi_1) \wedge Gr_{full}(\psi_2) & \text{if } \varphi := \psi_1 \wedge \psi_2 \\ Gr_{full}(\psi_1) \vee Gr_{full}(\psi_2) & \text{if } \varphi := \psi_1 \vee \psi_2 \\ \bigwedge_{d \in D} Gr_{full}(\psi[x/d]) & \text{if } \varphi := \forall x\ \psi[x] \\ \bigvee_{d \in D} Gr_{full}(\psi[x/d]) & \text{if } \varphi := \exists x\ \psi[x] \\ \{P(\overline{d}) \leftarrow Gr_{full}(\psi[\overline{x}/\overline{d}]) \mid \overline{d} \in D^n\} & \text{if } \varphi := \forall \overline{x}\ (P(\overline{x}) \leftarrow \psi) \end{cases} \tag{4}$$

*The full grounding of a definition* $\Delta$ *is the union of the full groundings of all rules in* $\Delta$*. The full grounding of a theory* $T$ *is the union of the full groundings of all sentences and definitions of* $T$*.*

We denote the full grounding for $\mathrm{MX}^\sigma_T(I_\sigma)$ by $\mathrm{Gr}_{\mathrm{full}}(T)$ if $\sigma$ and $I_\sigma$ are clear from the context. It is straightforward to show that $\mathrm{Gr}_{\mathrm{full}}(T)$ is indeed a grounding for $\mathrm{MX}^\sigma_T(I_\sigma)$.

An inductive definition like (4) can be evaluated in a top-down or bottom-up way. Both approaches are used in current grounders. On the one hand, there are grounders that go top-down through the syntax trees of the sentences in $T$. When a subformula $\varphi$ of the form $\forall x\ \psi[x]$, respectively $\exists x\ \psi[x]$ is reached, the grounding of $\psi[x/d]$ is constructed for every domain constant $d$, and then $\varphi$ is replaced by the conjunction, respectively disjunction, of all these groundings. GIDL [22] is an example of a top-down style grounder.

Other grounders go bottom-up through the syntax trees. For each subformula $\varphi[\overline{x}]$ a table is computed consisting of tuples $\overline{d}$ and corresponding groundings of $\varphi[\overline{x}/\overline{d}]$. These tables are computed first for atomic formulas and subsequently for compound formulas. For example, let $\varphi[x, y, z]$ be the formula $\psi[x, y] \wedge \chi[y, z]$ and assume the tables for $\psi$ and $\chi$ have been computed. Then the table for $\varphi$ is computed by taking the natural join of the tables for $\psi$ and $\chi$ on the value for $y$, and constructing the grounding for $\varphi[x/d_x, y/d_y, z/d_z]$ as the (possibly simplified) conjunction of the groundings for $\psi[x/d_x, y/d_y]$ and $\chi[y/d_y, z/d_z]$. Examples of bottom-up style grounders are KODKOD [19] and MXG [13].

To obtain a reduced grounding for $\mathrm{MX}^\sigma_T(I_\sigma)$ one could first construct the full grounding and then replace every subformula $\varphi$ over $\sigma$ in it by $\top$ if $I_\sigma \models \varphi$ and by $\bot$ otherwise. The result can be simplified further by recursively replacing $\bot \wedge \psi$ by $\bot$, $\top \wedge \psi$ by $\psi$, etc. The resulting grounding is the one computed by most current grounding algorithms and is often a lot smaller than the full grounding. We denote it by $\mathrm{Gr}_{\mathrm{red}}(T, \sigma, I_\sigma)$, or by $\mathrm{Gr}_{\mathrm{red}}(T)$ if $\sigma$ and $I_\sigma$ are clear from the context. Smart grounding algorithms try to avoid creating the full grounding by substituting ground formulas over the input vocabulary $\sigma$ as soon as possible. We refer the reader to [6, 14, 15, 18, 19] for an in-depth coverage of techniques to efficiently construct the reduced grounding $\mathrm{Gr}_{\mathrm{red}}(T)$.

## 3 Grounding with Bounds

We now present our method to add redundant information to an FO(ID) theory $T$ in order to optimize grounding size and time. The redundant information takes the form of a pair of bounds for each subformula of $T$. Each bound for a subformula $\varphi[\overline{x}]$ is a formula over the input vocabulary $\sigma$. It describes a set of tuples $\overline{d}$ for which $\varphi[\overline{x}/\overline{d}]$ is certainly true (false) in every solution of $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$. The larger the set described by a bound, the more precise the bound is.

First, we formally define bounds. Then, we indicate how bounds can be inserted in $T$ to obtain a new theory $T'$. The reduced grounding of $T'$ is often a lot smaller than the reduced grounding of $T$. The more precise the inserted bounds are, the smaller the grounding of $T'$ becomes. However, $T$ and $T'$ are in general not equivalent, since inserting bounds results in a weaker theory. I.e., the models of $T'$ form a superset of the models of $T$. Intuitively, the information that the inserted formulas are bounds, is lost in $T'$. To compensate for this lost information, sentences of the form $\forall\overline{x}\ (\psi \supset \varphi)$ should be added to $T'$. Such sentences state that $\psi$ is a bound for $\varphi$. Of course, adding these sentences again increases the grounding size. Hence, at first sight, we cannot be sure that inserting bounds yields smaller groundings of $T$. We solve this problem by defining a class of bounds that certainly yield smaller groundings.

### 3.1 Bounds

We distinguish between two kinds of bounds.

**Definition 5.** *A* certainly true bound (ct-bound) over $\sigma$ with respect to $T$ for a formula $\varphi[\overline{x}]$ *is a formula* $\varphi_{\mathrm{ct}}[\overline{y}]$ *over* $\sigma$ *such that* $\overline{y} \subseteq \overline{x}$ *and* $T \models \forall\overline{x}\ (\varphi_{\mathrm{ct}}[\overline{y}] \supset \varphi[\overline{x}])$. *Vice versa, a* certainly false bound (cf-bound) over $\sigma$ with respect to $T$ for $\varphi[\overline{x}]$ *is a formula* $\varphi_{\mathrm{cf}}[\overline{z}]$ *over* $\sigma$ *such that* $\overline{z} \subseteq \overline{x}$ *and* $T \models \forall\overline{x}\ (\varphi_{\mathrm{cf}}[\overline{z}] \supset \neg\varphi[\overline{x}])$.

We do not mention $\sigma$ and $T$ if they are clear from the context.

Intuitively, a ct-bound $\varphi_{\mathrm{ct}}$ for $\varphi[\overline{x}]$ provides for every structure $I_\sigma$ a lower bound for the set of tuples $\overline{d}$ for which $\varphi[\overline{x}/\overline{d}]$ is true in every solution of $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$. Vice versa, a cf-bound $\varphi_{\mathrm{cf}}$ provides a lower bound on the set of $\overline{d}$ for which $\varphi[\overline{x}/\overline{d}]$ is false. Observe that the negation of a ct-bound, respectively cf-bound, gives an upper bound on the set of tuples for which $\varphi$ is false, respectively true, in at least one solution of $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$.

*Example 3 (Example 1 ctd.).* Let $\varphi_1$ be the subformula $Sub(x,y) \wedge Sub(x,z)$ of $T_1$. Then $\neg Edge(x,y) \vee \neg Edge(x,z)$ is a cf-bound over $\sigma_1$ with respect to $T_1$ for $\varphi_1$. Indeed, one can derive from (1) that $T_1$ entails

$$\forall x \forall y \forall z\ \ ((\neg Edge(x,y) \vee \neg Edge(x,z)) \supset \neg\varphi_1)\,.$$

Observe that $\top$ is a ct-bound for every *sentence* of $T$. Also, $\bot$ is a ct-bound as well as a cf-bound for every *formula*. We call $\bot$ the *trivial bound*. Intuitively, the trivial bound contains no information at all. According to the following definition, it is the least precise bound.

**Definition 6.** *Let $\psi[\overline{y}]$ and $\chi[\overline{z}]$ be two (ct- or cf-) bounds for $\varphi[\overline{x}]$. We say that $\psi[\overline{y}]$* is more precise than *$\chi[\overline{z}]$ if $\forall \overline{x}\ (\chi[\overline{z}] \supset \psi[\overline{y}])$ is valid.*

If $\psi$ is a more precise bound for $\varphi[\overline{x}]$ than $\chi$, $\psi$ provides a larger lower bound.

**Definition 7.** *A* c-map *$\mathcal{C}$ for $T$ over $\sigma$ is a mapping from all subformulas $\varphi$ of $T$ to tuples $(\mathcal{C}^{\mathrm{ct}}(\varphi), \mathcal{C}^{\mathrm{cf}}(\varphi))$, where $\mathcal{C}^{\mathrm{ct}}(\varphi)$ and $\mathcal{C}^{\mathrm{cf}}(\varphi)$ are respectively a ct- and cf-bound for $\varphi$ over $\sigma$ with respect to $T$.*

The notion of precision pointwise extends to c-maps.

Let $M$ be a model of $T$ and $\mathcal{C}$ a c-map for $T$ over $\sigma$. From the definition of ct- and cf-bounds, it follows immediately that for every subformula $\varphi[\overline{x}]$ of $T$, $M$ satisfies both $\forall \overline{x}\ (\mathcal{C}^{\mathrm{ct}}(\varphi) \supset \varphi)$ and $\forall \overline{x}\ (\mathcal{C}^{\mathrm{cf}}(\varphi) \supset \neg\varphi)$. The set of all these sentences is denoted by $\overline{\mathcal{C}}$:

**Definition 8.** *Let $\mathcal{C}$ be a c-map for $T$ over $\sigma$. Then the theory $\overline{\mathcal{C}}$ is defined by*

$$\begin{aligned}\overline{\mathcal{C}} =& \{\forall \overline{x}\ (\mathcal{C}^{\mathrm{ct}}(\varphi) \supset \varphi) \mid \varphi[\overline{x}] \text{ is a subformula of } T\} \\ &\cup \{\forall \overline{x}\ (\mathcal{C}^{\mathrm{cf}}(\varphi) \supset \neg\varphi) \mid \varphi[\overline{x}] \text{ is a subformula of } T\}.\end{aligned}$$

A c-map $\mathcal{C}$ is *$I_\sigma$-inconsistent* if according to $\mathcal{C}$ and $I_\sigma$, some subformula $\varphi[\overline{x}]$ of $T$ is both certainly true and false for some tuple, i.e., $I_\sigma \models \exists \overline{x}\ (\mathcal{C}^{\mathrm{ct}}(\varphi) \wedge \mathcal{C}^{\mathrm{cf}}(\varphi))$. If there exists an $I_\sigma$-inconsistent c-map for $T$ over $\sigma$, then $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$ has no solutions.

### 3.2 C-Transformation

For the rest of this section, fix a c-map $\mathcal{C}$ for $T$ over $\sigma$. We now show how to insert the bounds of $\mathcal{C}$ into the sentences of $T$. The insertion consists of replacing every subformula $\varphi$ of $T$ by the new formula $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cf}}(\varphi)) \vee \mathcal{C}^{\mathrm{ct}}(\varphi)$. I.e., the new formula is true if $\varphi$ is certainly true according to $\mathcal{C}$, or if $\varphi$ is true and not certainly false according to $\mathcal{C}$. Observe that if $\mathcal{C}^{\mathrm{ct}}(\varphi) = \mathcal{C}^{\mathrm{cf}}(\varphi) = \bot$, then $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cf}}(\varphi)) \vee \mathcal{C}^{\mathrm{ct}}(\varphi)$ is logically equivalent to $\varphi$. As such, adding the trivial bounds to a formula $\varphi$ does not change the sentence at all.

**Definition 9 (c-transformation).** *A* c-transformation *of a subformula $\varphi$ of $T$ with respect to $\mathcal{C}$, denoted $\mathcal{C}\langle\varphi\rangle$, is the formula $(\varphi' \wedge \neg\mathcal{C}^{\mathrm{cf}}(\varphi)) \vee \mathcal{C}^{\mathrm{ct}}(\varphi)$ where $\varphi'$ is defined by*

$$\varphi' := \begin{cases} \varphi & \text{if } \varphi \text{ is an atom} \\ \neg\mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \neg\psi \\ \mathcal{C}\langle\psi\rangle \wedge \mathcal{C}\langle\chi\rangle & \text{if } \varphi \text{ is equal to } \psi \wedge \chi \\ \mathcal{C}\langle\psi\rangle \vee \mathcal{C}\langle\chi\rangle & \text{if } \varphi \text{ is equal to } \psi \vee \chi \\ \exists x\ \mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \exists x\ \psi \\ \forall x\ \mathcal{C}\langle\psi\rangle & \text{if } \varphi \text{ is equal to } \forall x\ \psi \end{cases}$$

*For a definition $\Delta$, $\mathcal{C}\langle\Delta\rangle$ is the definition $\bigcup_{\forall \overline{x}\ (P(\overline{x}) \leftarrow \varphi) \in \Delta} \forall \overline{x}\ (P(\overline{x}) \leftarrow \mathcal{C}\langle\varphi\rangle)$. A* c-transformation *$\mathcal{C}\langle T\rangle$ of $T$ with respect to $\mathcal{C}$ consists of a c-transformation with respect to $\mathcal{C}$ of every sentence and definition of $T$.*

One can prove that $\mathcal{C}\langle T\rangle$ is weaker than $T$, i.e., the models of $T$ are a subset of the models of $\mathcal{C}\langle T\rangle$. In general, $T$ and $\mathcal{C}\langle T\rangle$ are not equivalent. E.g., if $\mathcal{C}$ assigns $\top$ as ct-bound to every sentence of an FO theory $T$, then $\mathcal{C}\langle T\rangle$ is equivalent to $\top$, which is weaker than $T$. Intuitively, inserting bounds results in a loss of information, namely the information that the inserted formulas are indeed bounds. If $T$ is an FO theory, this information is captured by the theory $\overline{\mathcal{C}}$ of Definition 8: it can be shown that $T$ and $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$ are equivalent. However, this property does not hold for definitions: in general $\mathcal{C}\langle \Delta\rangle \cup \overline{\mathcal{C}}$ is not equivalent to $\Delta \cup \overline{\mathcal{C}}$ for a definition $\Delta$. Sufficient conditions on $\mathcal{C}$ to nevertheless ensure equivalence are given in the following proposition. Here we say that a formula occurs positively (negatively) in a definition $\Delta$ if it occurs in the scope of an even (odd) number of negations in a body of a rule in $\Delta$.

**Proposition 1.** *Let $\mathcal{C}$ be a c-map and $\Delta$ a definition. Then $\mathcal{C}\langle \Delta\rangle \cup \overline{\mathcal{C}}$ is equivalent to $\Delta \cup \overline{\mathcal{C}}$ if for every subformula $\varphi$ of $\Delta$ that contains a predicate $P \in Def(\Delta)$, the following hold:*

1. *If $\Delta$ is not total, then $\mathcal{C}^{\mathrm{ct}}(\varphi) = \mathcal{C}^{\mathrm{cf}}(\varphi) = \bot$.*
2. *If $\varphi$ occurs positively in $\Delta$ and $P$ occurs positively in $\varphi$, then $\mathcal{C}^{\mathrm{ct}}(\varphi) = \bot$.*
3. *If $\varphi$ occurs negatively in $\Delta$ and $P$ occurs negatively in $\varphi$, then $\mathcal{C}^{\mathrm{cf}}(\varphi) = \bot$.*

A c-map for $T$ that satisfies the conditions of Proposition 1 for every definition $\Delta$ of $T$ is called *$T$-tolerant*. Now we can formulate our main theorem.

**Theorem 1.** *If $\mathcal{C}$ is $T$-tolerant, then $T$ and $\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}$ are equivalent.*

**Corollary 1.** *If $\mathcal{C}$ is $T$-tolerant, then $\mathrm{MX}^{\sigma}_{T}(I_\sigma) = \mathrm{MX}^{\sigma}_{\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}}(I_\sigma)$ for any $\sigma$-structure $I_\sigma$.*

### 3.3 Atom-Based and Atom-Equal C-Maps

Corollary 1 states that we can solve $\mathrm{MX}^{\sigma}_{T}$ for input $I_\sigma$ by first computing a c-map $\mathcal{C}$ for $T$ over $\sigma$ and then solving $\mathrm{MX}^{\sigma}_{\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}}(I_\sigma)$. This approach is beneficial if $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}})$ is smaller than $\mathrm{Gr}_{\mathrm{red}}(T)$, and can be constructed at least as fast. In general, these conditions are not satisfied. The more precise c-map $\mathcal{C}$ is, the smaller the reduced grounding $\mathcal{C}\langle T\rangle$ becomes, but the larger the reduced grounding of $\overline{\mathcal{C}}$ is. A c-map that is useful to reduce grounding size should therefore not be too precise, in order to avoid a blow-up of $\mathrm{Gr}_{\mathrm{red}}(\overline{\mathcal{C}})$, but still be precise enough to decrease the size of $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle)$.

A sufficient condition to obtain small groundings is to use an *atom-based, atom-equal* c-map $\mathcal{C}$. We say that $\mathcal{C}$ is atom-based if $\overline{\mathcal{C}}_A \models \overline{\mathcal{C}}$, where

$$\begin{aligned}\overline{\mathcal{C}}_A = &\{\forall \overline{x}\ (\mathcal{C}^{\mathrm{ct}}(\varphi) \supset \varphi)\ \mid\ \varphi[\overline{x}] \text{ is an atomic subformula of } T\}\\ &\cup \{\forall \overline{x}\ (\mathcal{C}^{\mathrm{cf}}(\varphi) \supset \neg\varphi)\ \mid\ \varphi[\overline{x}] \text{ is an atomic subformula of } T\}.\end{aligned}$$

Intuitively, $\mathcal{C}$ is atom-based if only information about bounds of atomic subformulas is lost in $\mathcal{C}\langle T\rangle$. A c-map $\mathcal{C}$ is atom-equal if it assigns essentially the same bounds to all atomic subformulas over the same predicate:

**Definition 10.** *A c-map $\mathcal{C}$ for $T$ over $\sigma$ is* atom-equal *if for all atomic subformulas $\varphi[\overline{x}]$ and $\psi[\overline{y}]$ of $T$ and all tuples $\overline{z}_x$ and $\overline{z}_y$ of variables such that $\varphi[\overline{x}/\overline{z}_x] = \psi[\overline{y}/\overline{z}_y]$, it holds that both $\mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{z}_x] \equiv \mathcal{C}^{\mathrm{ct}}(\psi)[\overline{y}/\overline{z}_y]$ and $\mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{z}_x] \equiv \mathcal{C}^{\mathrm{cf}}(\psi)[\overline{y}/\overline{z}_y]$ are valid.*

The following proposition states the desired property of atom-based, atom-equal c-maps:

**Proposition 2.** *Let $\mathcal{C}$ be an atom-based, atom-equal c-map. If $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$ has a solution, then $Gr_{red}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A)$ is smaller than $Gr_{red}(T)$.*

From this proposition, we derive the following grounding algorithm to create a small grounding for $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$.

1. Compute a $T$-tolerant, atom-based, atom-equal c-map $\mathcal{C}$ for $T$ over $\sigma$.
2. If $\mathcal{C}$ is $I_\sigma$-inconsistent, output $\bot$ and stop.
3. Else, output $\mathrm{Gr}_{\mathrm{red}}(\mathcal{C}\langle T\rangle \cup \overline{\mathcal{C}}_A, \sigma, I_\sigma)$, using an off-the-shelf grounding algorithm.

It follows from Theorem 1 and the definition of atom-based that the result of this algorithm is indeed a grounding for $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$. Observe that the first step of this algorithm is independent of $I_\sigma$. If one has to solve $\mathrm{MX}^{\sigma}_{T}$ for multiple inputs $I_\sigma$, then it suffices to compute $\mathcal{C}$ only once.

### 3.4 Computing Bounds

An algorithm to compute an atom-based, atom-equal c-map for an FO theory was presented in [23]. We now extend this algorithm to FO(ID). The *completion* of a definition $\Delta$ is the FO theory that contains for every $P \in \mathrm{Def}(\Delta)$ the sentence $\forall \overline{x}\ (P(\overline{x}) \equiv ((\overline{x} = \overline{y}_1 \wedge \varphi_1) \vee \ldots \vee (\overline{x} = \overline{y}_n \wedge \varphi_n)))$, where $\forall \overline{y}_1\ P(\overline{y}_1) \leftarrow \varphi_1$, $\ldots$, $\forall \overline{y}_n\ P(\overline{y}_n) \leftarrow \varphi_n$ are all rules in $\Delta$ with $P$ in the head. Clearly, every subformula of $\Delta$ occurs in the completion of $\Delta$. If $T$ is an FO(ID) theory then we denote by $\mathrm{Comp}(T)$ the FO theory obtained by replacing in $T$ all definitions by their completion. The theory $\mathrm{Comp}(T)$ is weaker than $T$ [4].

Our method to compute a c-map $T$ works in three stages. First construct $\mathrm{Comp}(T)$. Then compute a c-map $\mathcal{C}$ for $\mathrm{Comp}(T)$ using the algorithm of [23]. Finally, assign to every subformula of $T$ the bounds assigned by $\mathcal{C}$ to the corresponding subformula in $\mathrm{Comp}(T)$. Denote the resulting c-map by $\mathcal{C}'$. Because $\mathrm{Comp}(T)$ is weaker than $T$, $\mathcal{C}'$ is indeed a c-map for $T$. To transform it to a $T$-tolerant c-map, it suffices to replace some of the bounds assigned by $\mathcal{C}'$ by $\bot$, as indicated by Proposition 1.

In general, directly using a c-map computed by the algorithm of [23] results in adding the same redundant information several times to a formula. E.g., the c-map $\mathcal{C}$ computed for theory $T_1$ of Example 1 assigns

$$\mathcal{C}^{\mathrm{ct}}(Sub(x,y)) = \neg Edge(x,y)$$
$$\mathcal{C}^{\mathrm{cf}}(Sub(x,z)) = \neg Edge(x,z)$$
$$\mathcal{C}^{\mathrm{cf}}(Sub(x,y) \wedge Sub(x,z)) = \neg(Edge(x,y) \wedge Edge(x,z)).$$

For this c-map, the c-transformation of $Sub(x, y) \wedge Sub(x, z)$ is given by

$$((Sub(x, y) \wedge Edge(x, y)) \wedge (Sub(x, z) \wedge Edge(x, z))) \wedge (Edge(x, y) \wedge Edge(x, z)).$$

This formula contains repeated constraints $Edge(x, y)$ and $Edge(x, z)$ on the variables $x$, $y$, and $z$. These could easily be eliminated, but it depends on the used grounding algorithm which ones are best deleted. In the next section, we indicate which bounds are deleted by the grounder GIDL.

## 4 Implementation and Experiments

In this section, we present a grounding algorithm based on the above theoretical results. In particular, we show how bounds can be incorporated in a simple "top-down style" grounder (see Section 2.2) without explicitly constructing $\mathcal{C}\langle T\rangle$. We report on our implementation, called GIDL, of the grounding algorithm and on experiments comparing naive grounding to grounding with bounds.

### 4.1 Grounding with Bounds

For the rest of this section, let $\mathcal{C}$ be an $I_\sigma$-consistent, $T$-tolerant, atom-based c-map for $T$ over $\sigma$. We call a formula of the form $\varphi \vee \psi$ or $\exists x\ \varphi$ a *disjunctive formula*. Vice versa, a *conjunctive formula* is of the form $\varphi \wedge \psi$ or $\forall x\ \varphi$.

**Grounding** Algorithm 1 shows a grounding algorithm that uses bounds, but does not construct $\mathcal{C}\langle T\rangle$ explicitly. Basically, it consults the bounds assigned by $\mathcal{C}$ whenever it substitutes the free variables of a formula $\varphi[\overline{x}]$ by domain constants $\overline{d}$. If according to the bounds, $\varphi[\overline{x}/\overline{d}]$ is certainly true, i.e., $I_\sigma \models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$, then the grounding of $\varphi[\overline{x}/\overline{d}]$ is not computed. Instead, the algorithm then proceeds as if $\varphi[\overline{x}/\overline{d}]$ is equal to $\top$. Similarly if $\varphi[\overline{x}/\overline{d}]$ is certainly false.

In line 1 of Algorithm 1, it is checked whether one of the sentences of $T$ is certainly false. If this is the case, then clearly $\mathrm{MX}_T^\sigma$ is unsatisfiable, and this can be reported immediately. Before a sentence is grounded, it is checked in line 4 whether this sentence is certainly true according to $\mathcal{C}$. Only sentences that are not certainly true are grounded. Observe that both checks are simple syntactic checks and can be executed in constant time.

Function `groundConj` gets as input a formula $\varphi[\overline{x}]$ and returns a grounding for $\forall\overline{x}\ \varphi[\overline{x}]$. In particular, if $\varphi$ is a sentence, then the result of applying `groundConj` to $\varphi$ is a grounding for $\varphi$. In `groundConj`, universal quantifiers are implicitly pushed inside conjunctions. I.e., if $\varphi[\overline{x}]$ is a conjunction $\psi_1 \wedge \ldots \wedge \psi_n$, then for every $i \in [1, n]$, the grounding of $\forall\overline{x}\ \psi_i$ is computed by applying `groundConj` to $\psi_i$. The conjunction of these groundings is returned as grounding for $\forall\overline{x}\ \varphi$.

Function `groundConj` only consults the c-map when variables are substituted by domain constants or when the input formula is an atom. As such, `groundConj` ignores ("eliminates") the bounds assigned to conjunctive formulas. As we mentioned at the end of Section 3.4, this is important to avoid repeated constraints on a variable.

**Algorithm 1**: Ground with Bounds

**Input**: $T$, $\sigma$, $I_\sigma$ and $\mathcal{C}$
**Output**: A grounding $T_g$ for $\mathrm{MX}^{\sigma}_{T}(I_\sigma)$

**if** $\mathcal{C}^{\mathrm{cf}}(\varphi) = \top$ *for some sentence* $\varphi$ *of* $T$ **then return** $\bot$;
$T_g := \emptyset$;
`// Ground all sentences of` $T$
**for** *every sentence* $\varphi$ *of* $T$ **do**
  **if** $\mathcal{C}^{\mathrm{ct}}(\varphi) \neq \top$ **then** Add `groundConj(`$\varphi$`)` to $T_g$;
`// Ground all definitions of` $T$
**for** *every definition* $\Delta$ *of* $T$ **do**
  Add `groundDef(`$\Delta$`)` to $T_g$;
`// Add the grounding of` $\overline{\mathcal{C}}_A$
**for** *every atomic subformula* $\varphi[\overline{x}]$ *of* $T$ **do**
  **for** *every* $\overline{d}$ *such that* $I_\sigma \models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ **do**
    Add $\varphi[\overline{x}/\overline{d}]$ to $T_g$;
  **for** *every* $\overline{d}$ *such that* $I_\sigma \models \mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{d}]$ **do**
    Add $\neg\varphi[\overline{x}/\overline{d}]$ to $T_g$;
**return** $T_g$;

**Function** `groundConj(`$\varphi[\overline{x}]$`)`

$C := \emptyset$;
**switch** $\varphi[\overline{x}]$ **do**
  **case** $\varphi$ *is a literal*
    **for** *all* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ **do**
      **if** $I_\sigma \models \mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{d}]$ **then return** $\bot$;
      **else** Add $\varphi[\overline{x}/\overline{d}]$ to $C$;
  **case** $\varphi = \forall y\ \psi[\overline{x}, y]$
    **return** `groundConj(`$\psi[\overline{x}, y]$`)`;
  **case** $\varphi = \bigwedge_i \psi_i$
    **return** $\bigwedge_i$ `groundConj(`$\psi_i$`)`;
  **case** $\varphi$ *is a disjunctive formula*
    **for** *all* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ **do**
      **if** $I_\sigma \models \mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{d}]$ **then return** $\bot$;
      **else** Add `groundDisj(`$\varphi[\overline{x}/\overline{d}]$`)` to $C$;
**return** $\bigwedge C$;

**Function** `groundDisj(`$\varphi[\overline{x}]$`)`

$D := \emptyset$;
**switch** $\varphi[\overline{x}]$ **do**
  **case** $\varphi$ *is a literal*
    **for** *all* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{d}]$ **do**
      **if** $I_\sigma \models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ **then return** $\top$;
      **else** Add $\varphi[\overline{x}/\overline{d}]$ to $D$;
  **case** $\varphi = \exists y\ \psi[\overline{x}, y]$
    **return** `groundDisj(`$\psi[\overline{x}, y]$`)`;
  **case** $\varphi = \bigvee_i \psi_i$
    **return** $\bigvee_i$ `groundDisj(`$\psi_i$`)`;
  **case** $\varphi$ *is a conjunctive formula*
    **for** *all* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{cf}}(\varphi)[\overline{x}/\overline{d}]$ **do**
      **if** $I_\sigma \models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ **then return** $\top$;
      **else** Add `groundConj(`$\varphi[\overline{x}/\overline{d}]$`)` to $D$;
**return** $\bigvee D$;

**Function** `groundDef(`$\Delta$`)`

$\Delta_g := \emptyset$;
**for** *every rule* $\forall \overline{x}\ P(\overline{x}) \leftarrow \varphi[\overline{y}]$ *in* $\Delta$ **do**
  $\overline{z} := \overline{x} \setminus \overline{y}$;
  **for** *every* $\overline{d}$ *such that* $I_\sigma \not\models \mathcal{C}^{\mathrm{cf}}(\varphi[\overline{y}/\overline{d}])$ **do**
    **if** $I_\sigma \models \mathcal{C}^{\mathrm{ct}}(\varphi[\overline{y}/\overline{d}])$ **then** $\varphi_g := \top$;
    **else** $\varphi_g :=$ `groundConj(`$\varphi[\overline{y}/\overline{d}]$`)`;
    Add $P(\overline{x})[\overline{y}/\overline{d}, \overline{z}/\overline{d}'] \leftarrow \varphi_g$ to $\Delta_g$ for every $\overline{d}'$;
**return** $\Delta_g$;

In `groundConj`$(\varphi[\overline{x}])$, only those substitutions $\varphi[\overline{x}/\overline{d}]$ are grounded for which $I_\sigma \not\models \mathcal{C}^{\mathrm{ct}}(\varphi)[\overline{x}/\overline{d}]$ (see, e.g., line 12). Indeed, the other substitutions yield a formula that is certainly true in all solutions of $\mathrm{MX}^\sigma_T(I_\sigma)$, and can therefore be omitted from the ground conjunction $C$ that is computed. Before $\varphi[\overline{x}/\overline{d}]$ is grounded, it is checked whether this substitution yields a formula that is certainly false (see, e.g., line 13). If this is the case, the whole conjunction $C$ will certainly be false, and therefore $\bot$ is returned immediately. Observe that implicitly the formula $(\varphi \wedge \neg\mathcal{C}^{\mathrm{cf}}(\varphi)) \vee \mathcal{C}^{\mathrm{ct}}(\varphi)$ is grounded. Hence the correctness of `groundConj` follows from Theorem 1.

Function `groundDisj` is dual to `groundConj`. On input $\varphi[\overline{x}]$, it returns a grounding for $\exists\overline{x}\ \varphi$. It implicitly pushes existential quantifiers through disjunctions and eliminates the bounds assigned to disjunctive formulas. Function `groundDef` returns a grounding for its input definition $\Delta$. It grounds the rules of $\Delta$ one-by-one. For each rule $\forall\overline{x}\ P(\overline{x}) \leftarrow \varphi[\overline{y}]$, only those substitutions $\varphi[\overline{y}/\overline{d}]$ that are possibly true, are tried (line 4). If $\varphi[\overline{y}/\overline{d}]$ is certainly true, it is replaced by $\top$ (line 5).

Clearly, the complexity of Algorithm 1 critically depends on the complexity of computing the truth value in $I_\sigma$ of (some of the) bounds assigned by $\mathcal{C}$. If the bounds become too complex, the time and space needed to evaluate them in $I_\sigma$ may exceed the time and space needed for constructing the full grounding and simplifying it afterwards. In GIDL, the complexity of bounds is estimated and too complex bounds are discarded.

### 4.2 Experiments

Algorithm 1 is implemented in the grounder GIDL. Besides FO(ID) (with functions), GIDL supports multi-sorted FO(ID), partial functions, arithmetic and aggregates [24]. Experiments comparing GIDL with other grounders were presented in [22].

In this section, we present results showing the impact of grounding with bounds. As input for GIDL, we used 37 benchmark problems, mainly taken from [1]. The used encodings and details about the experiments are available at `www.cs.kuleuven.be/~dtai/krr/software.html`. We used three different versions of GIDL:

- GIDL$_{nb}$**:** Assigns $\langle\varphi, \neg\varphi\rangle$ as bound to every atomic subformula $\varphi$ over the input vocabulary, and $\langle\bot, \bot\rangle$ to every other subformula. As such, it creates the reduced grounding of the input theory.
- GIDL$_{mn}$ **and** GIDL$_r$**:** Compute a c-map as described in Section 3.4 and delete bounds that are too complex. In GIDL$_{mn}$, the complexity of a formula is dictated by its length. GIDL$_r$ estimates the reward vs. cost of using a bound and removes bounds with high cost and small reward.

In Table 1, the influence of bounds on the grounding size and time is shown. The second and third column show the ratio of the grounding size obtained with GIDL$_{mn}$ and GIDL$_r$ compared to $\mathrm{Gr}_{\mathrm{red}}(T)$. When interpreting Table 1, it is

important to note that small reductions in grounding size are not important. The reason being that all reductions that can be obtained by the using bounds are also obtained by applying unit propagation on the grounding. Since there exist very efficient implementations of unit propagation, it is not beneficial to let the refinement algorithm find small reductions at a relatively high cost. We see that both GIDL$_{mn}$ and GIDL$_r$ reduce the grounding size with more than 50% in around 30% of the benchmarks. In 7, respectively 6, benchmarks there is a spectacular reduction of more than 95%.

More important than reductions in size are reductions in grounding time. Columns 4-6 show the running times of the different versions of GIDL, and (between brackets) the ratio of the running time to the running time of GIDL$_{nb}$. The time to compute the c-map is included (it never took more than 0.02 seconds). A time-out (###) of 600 seconds was used.

From Table 1, we can see that GIDL$_{mn}$ performs quite well. On half of the benchmarks, it is more than 44% faster than GIDL$_{nb}$. There are some outliers however. On benchmarks 6 it is far slower than GIDL$_{nb}$, while not producing a significantly smaller grounding. On benchmark 11, it is more than six times slower than GIDL$_r$. This indicates the use of a complex bound with relatively small reward. Compared to GIDL$_{mn}$, GIDL$_r$ is faster and more robust, indicating that using estimators for the reward and cost of bounds pays off in most cases. In only two of the benchmarks, the estimators make a wrong guess. In benchmark 1, a bound with high cost and no reward is allowed, in benchmark 7, a bound with low cost and high reward is not allowed by GIDL$_r$. It is part of future work to implement less naive estimators.

On many benchmarks, the reduction in grounding time with respect to GIDL$_{nb}$ is due to the reduction in grounding size. Yet there are also several benchmarks where time decreases a lot, while there is almost no reduction in size. The reason is that a c-map spreads redundant information over the whole theory, which allows for earlier pruning by a top-down style grounder, and hence faster grounding.

We conclude that grounding with bounds is applicable in practice. It often leads to smaller grounding sizes on standard benchmark problems, and if the bounds are carefully restricted, it yields a significant speed up. For an explanation why these smaller groundings do not degrade the performance of subsequently applied propositional model generators (SAT solvers), we refer the reader to [21], Appendix B.

## 5 Conclusions

In this paper, we presented an automated method to add redundant information to FO(ID) theories. This redundant information is useful to efficiently create smaller groundings. The method can be used as a preprocessor for any grounder or can be integrated in a grounding algorithm. We presented such an integration for a top-down style grounding algorithm. Experiments show that adding the redundant information leads to smaller grounding sizes and times.

**Table 1.** Impact of bounds on grounding

| | | Size | | Time | | | |
|---|---|---|---|---|---|---|---|
| Nr | Benchmark name | GIDL$_{mn}$ | GIDL$_{r}$ | GIDL$_{nb}$ | GIDL$_{mn}$ | | GIDL$_{r}$ |
| 1 | 15-puzzle | 1.00 | 1.00 | 6.13 | 2.07 | (0.34) | 5.73 (0.93) |
| 2 | Battleship | 0.89 | 1.00 | 0.19 | 0.16 | (0.84) | 0.17 (0.89) |
| 3 | Blocked N-queens | 0.02 | 0.02 | 9.66 | 2.22 | (0.23) | 2.67 (0.28) |
| 4 | Blocks world | 0.33 | 0.33 | 22.33 | 5.80 | (0.26) | 5.80 (0.26) |
| 5 | Bounded spanning tree | 0.12 | 0.12 | 8.52 | 3.01 | (0.35) | 1.16 (0.14) |
| 6 | Clique | 1.00 | 1.00 | 3.13 | 51.77 | (16.54) | 3.73 (1.19) |
| 7 | Hierarchical clustering | 0.03 | 0.72 | 0.32 | 0.05 | (0.16) | 0.31 (0.97) |
| 8 | Graph colouring | 1.00 | 1.00 | 2.57 | 2.69 | (1.05) | 2.72 (1.06) |
| 9 | Debugging | 0.86 | 1.00 | 0.30 | 0.48 | (1.60) | 0.47 (1.57) |
| 10 | Fastfood | 1.00 | 1.00 | ### | 17.59 | (0.03) | 16.52 (0.03) |
| 11 | FO-hamcircuit | 0.94 | 0.99 | ### | 37.86 | (0.06) | 6.06 (0.01) |
| 12 | Golomb ruler | 0.54 | 1.00 | 14.05 | 4.13 | (0.29) | 3.40 (0.24) |
| 13 | Graph partitioning | 0.94 | 1.00 | 0.03 | 0.03 | (1.00) | 0.02 (0.67) |
| 14 | Algebraic groups | 0.99 | 1.00 | 9.68 | 11.20 | (1.16) | 9.60 (0.99) |
| 15 | Hamiltonian circuit | 0.01 | 0.01 | 70.75 | 2.56 | (0.04) | 1.81 (0.03) |
| 16 | Tower of Hanoi | 1.00 | 1.00 | 2.32 | 1.96 | (0.84) | 1.83 (0.79) |
| 17 | Knight tour | 0.00 | 0.00 | 12.22 | 0.06 | (0.00) | 0.10 (0.01) |
| 18 | Labyrinth | 0.99 | 0.99 | 8.80 | 8.83 | (1.00) | 8.73 (0.99) |
| 19 | Magic series | 1.00 | 1.00 | 1.83 | 1.79 | (0.98) | 1.81 (0.99) |
| 20 | Maze generation | 0.90 | 0.90 | 2.77 | 0.51 | (0.18) | 0.17 (0.06) |
| 21 | Mirror puzzle | 1.00 | 1.00 | 0.12 | 0.12 | (1.00) | 0.10 (0.83) |
| 22 | Missionaries | 0.03 | 0.03 | 17.4 | 2.29 | (0.13) | 2.68 (0.15) |
| 23 | N-queens | 1.00 | 1.00 | 4.62 | 4.62 | (1.00) | 4.64 (1.00) |
| 24 | Pigeon hole | 1.00 | 1.00 | 4.92 | 4.90 | (1.00) | 4.90 (1.00) |
| 25 | Disjunctive scheduling | 0.83 | 0.83 | 151.15 | 172.50 | (1.14) | 171.54 (1.13) |
| 26 | Slitherlink | 0.04 | 0.04 | 0.25 | 0.02 | (0.08) | 0.02 (0.08) |
| 27 | Social golfer | 1.00 | 1.00 | 5.47 | 5.37 | (0.98) | 5.41 (0.99) |
| 28 | Sokoban | 0.59 | 0.59 | 2.78 | 1.57 | (0.56) | 1.54 (0.55) |
| 29 | Solitaire | 1.00 | 0.73 | 0.43 | 0.46 | (1.07) | 0.49 (1.14) |
| 30 | Spanning tree | 0.06 | 0.06 | 6.86 | 0.59 | (0.09) | 0.57 (0.08) |
| 31 | Sudoku | 0.75 | 0.75 | ### | 1.07 | (0.00) | 1.06 (0.00) |
| 32 | Tarski | 1.00 | 1.00 | 4.42 | 3.67 | (0.83) | 3.64 (0.82) |
| 33 | Tough nut | 0.00 | 0.00 | 4.23 | 0.53 | (0.13) | 0.53 (0.13) |
| 34 | Train scheduling | 0.25 | 0.25 | 4.06 | 0.65 | (0.16) | 0.47 (0.12) |
| 35 | Water bucket | 0.36 | 0.36 | 3.16 | 1.76 | (0.56) | 2.04 (0.65) |
| 36 | Bounded dominating set | 1.00 | 1.00 | 1.45 | 0.03 | (0.02) | 0.03 (0.02) |
| 37 | Wire routing | 0.92 | 0.99 | 0.06 | 0.08 | (1.33) | 0.08 (1.33) |
| | Total | | | 2186.98 | 355.00 | (0.16) | 272.55 (0.12) |
| | Average gain | 34 % | 30 % | | | 0 % | 40 % |
| | Median gain | 17 % | 1 % | | | 44 % | 40 % |
| | # < 1.00 | 24 | 20 | | | 25 | 29 |
| | # < 0.50 | 12 | 11 | | | 18 | 16 |
| | # < 0.05 | 7 | 6 | | | 5 | 6 |

In the future, we plan to extend our method to other extensions of FO. Also, it would be interesting to see the impact of integrating bounds in bottom-up style grounders such as MXG [13] or KODKOD [19].

## References


1. Asparagus. `http://asparagus.cs.uni-potsdam.de/`.
2. Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style model finding. In *MODEL*, 2003.
3. Marc Denecker. Extending classical logic with inductive definitions. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *CL*, volume 1861 of *Lecture Notes in Computer Science*, pages 703–717. Springer, 2000.
4. Marc Denecker and Eugenia Ternovska. A logic of nonmonotone inductive definitions. *ACM Transactions on Computational Logic (TOCL)*, 9(2):Article 14, 2008.
5. Marc Denecker and Joost Vennekens. Building a knowledge base system for an integration of logic programming and classical logic. In María García de la Banda and Enrico Pontelli, editors, *ICLP*, volume 5366 of *Lecture Notes in Computer Science*, pages 71–76. Springer, 2008.
6. Deborah East, Mikhail Iakhiaev, Artur Mikitiuk, and Mirosław Truszczyński. Tools for modeling and solving search problems. *AI Communications*, 19(4):301–312, 2006.
7. Herbert B. Enderton. *A Mathematical Introduction To Logic*. Academic Press, 1972.
8. Martin Gebser, Torsten Schaub, and Sven Thiele. GrinGo : A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.
9. Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *AAAI/IAAI*, pages 1194–1201. AAAI Press, 1996.
10. Mark-A. Krogel, Simon Rawles, Filip Zelezný, Peter A. Flach, Nada Lavrac, and Stefan Wrobel. Comparative evaluation of approaches to propositionalization. In Tamás Horváth, editor, *ILP*, volume 2835 of *Lecture Notes in Computer Science*, pages 197–214. Springer, 2003.
11. Maarten Mariën, David Gilis, and Marc Denecker. On the relation between ID-Logic and Answer Set Programming. In José Júlio Alferes and João Alexandre Leite, editors, *JELIA*, volume 3229 of *Lecture Notes in Computer Science*, pages 108–120. Springer, 2004.
12. William McCune. Mace4 reference manual and guide. *CoRR*, cs.SC/0310055, 2003.
13. David G. Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebali. Model expansion as a framework for modelling and solving search problems. Technical Report TR 2006-24, Simon Fraser University, Canada, 2006.
14. Murray Patterson, Yongmei Liu, Eugenia Ternovska, and Arvind Gupta. Grounding for model expansion in k-guarded formulas with inductive definitions. In Manuela M. Veloso, editor, *IJCAI*, pages 161–166, 2007.
15. Simona Perri, Francesco Scarcello, Gelsomina Catalano, and Nicola Leone. Enhancing DLV instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):195–228, 2007.

16. Deepak Ramachandran and Eyal Amir. Compact propositional encodings of first-order theories. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 340–345. AAAI Press / The MIT Press, 2005.
17. Stephan Schulz. A comparison of different techniques for grounding near-propositional cnf formulae. In Susan M. Haller and Gene Simmons, editors, *FLAIRS Conference*, pages 72–76. AAAI Press, 2002.
18. Tommi Syrjänen. *Logic Programs and Cardinality Constraints: Theory and Practice*. Doctoral dissertation, TKK Dissertations in Information and Computer Science TKK-ICS-D12, Helsinki University of Technology, Faculty of Information and Natural Sciences, Department of Information and Computer Science, Espoo, Finland, 2009.
19. Emina Torlak and Daniel Jackson. Kodkod: A relational model finder. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 632–647. Springer, 2007.
20. Allen Van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
21. Johan Wittocx. *Finite Domain and Symbolic Inference Methods for Extensions of First-Order Logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, May 2010.
22. Johan Wittocx, Maarten Mariën, and Marc Denecker. GIDL: A grounder for $FO^+$. In Maurice Pagnucco and Michael Thielscher, editors, *NMR*, pages 189–198. University of New South Wales, 2008.
23. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding with bounds. In Dieter Fox and Carla P. Gomes, editors, *AAAI*, pages 572–577. AAAI Press, 2008.
24. Johan Wittocx, Maarten Mariën, and Marc Denecker. The IDP system: a model expansion system for an extension of classical logic. In Marc Denecker, editor, *LaSh*, pages 153–165, 2008.
25. Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *Journal of Artificial Intelligence Research*, 2010. accepted.